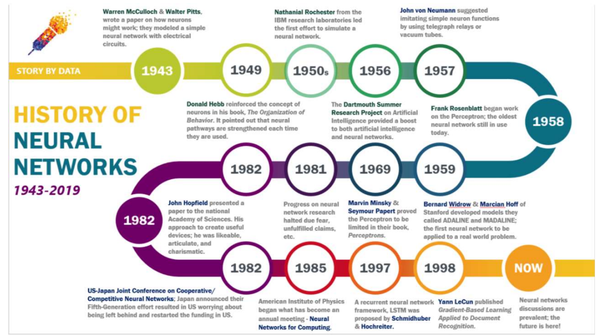
Unit – III: NEURAL NETWORKS

- Characteristics of Neural Networks, Historical Development of Neural Networks Principles. Artificial Neural Networks: Terminology, Models of Neuron, Topology, Basic Learning Laws, Pattern Recognition Problem, Basic Functional Units, Pattern Recognition Tasks by the Functional Units.

Historical Development of Neural Networks



Artificial Intelligence, Machine Learning and Deep Learning

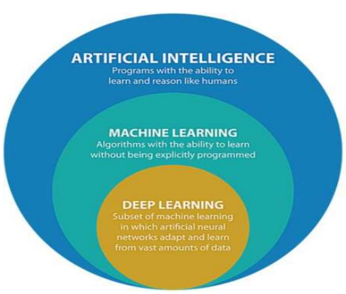
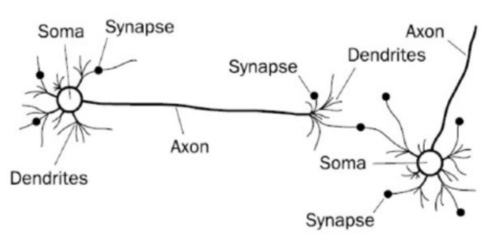
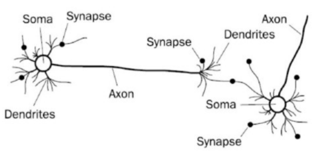


Image Source: https://datacatchup.com/artificial-intelligence-machine-learning-and-deep-learning/

Biological Neural Networks

- A nerve cell neuron is a special biological cell that processes information.





Dendrites − They are tree-like branches, responsible for receiving the information from other neurons it is connected to. In other sense, we can say that they are like the ears of neuron.

Soma − It is the cell body of the neuron and is responsible for processing of information, they have received from dendrites.

Axon − It is just like a cable through which neurons send the information.

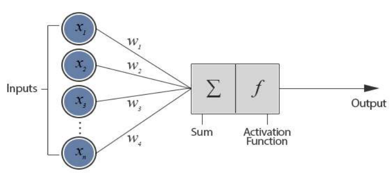
Synapses − It is the connection between the axon and other neuron dendrites.

Characteristic of Biological Neuron

1) Non-linearity
2) Input-output Mapping
3) Adaptability
4) Evidential Response (degree of 'confidence)
5) Fault Tolerance
6) Massively Parallel Computing

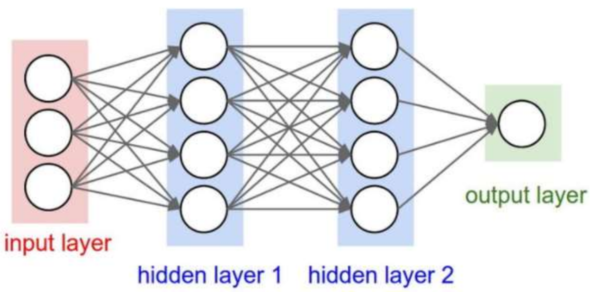
Artificial Neural Networks (ANN)

- Artificial neural networks, usually simply called neural networks, are **computing systems (mathematical function)**, inspired by the **biological neural networks.**
- **Artificial neurons** are elementary units in an **artificial neural** network.
- The weights can either amplify or deamplify the original input signal. For example, if the input is 1 and the input's weight is 0.2 the input will be decreased to 0.2.



Most artificial neurons have three things –
- an input transformation function
- an activation function
- an output transformation function

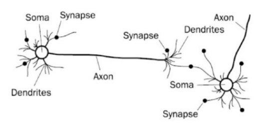
- Input transformation functions process the incoming signal – usually multiply and sum Artificial Neurons
- Activation functions process the input signal
  - Threshold Function
  - Linear
  - Saturated linear
  - Sigmoid
- Output functions process the outgoing signal – usually linear

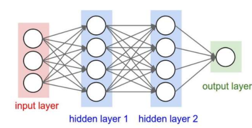


https://www.digitaltrends.com/cool-tech/what-is-an-artificial-neural-network/

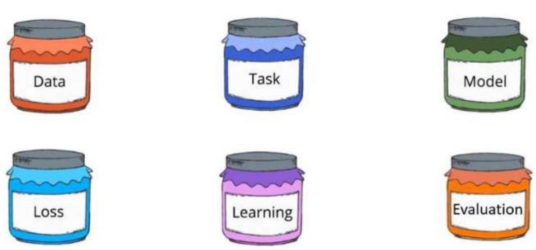
Biological Neural Network and Artificial Neural Network ANN

Biological Neural Network BNN	Artificial Neural Network ANN
Soma	Node
Dendrites	Input
Synapse	Weights or Interconnections
Axon	Output

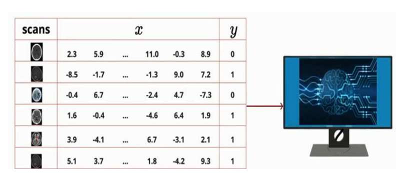




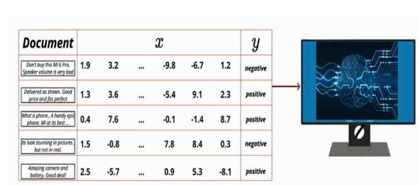
Six Jars of Deep Learning



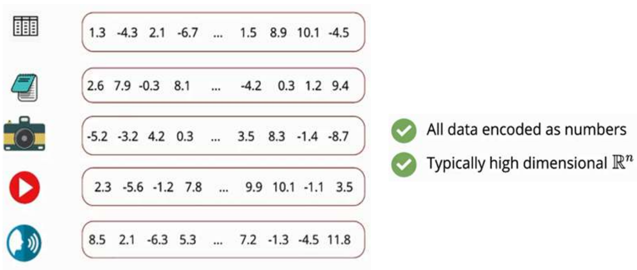
Data- Data everywhere



scans	x						y
	2.3	5.9	...	11.0	-0.3	8.9	0
	-8.5	-1.7	...	-1.3	9.0	7.2	1
	-0.4	6.7	...	-2.4	4.7	-7.3	0
	1.6	-0.4	...	-4.6	6.4	1.9	1
	3.9	-4.1	...	6.7	-3.1	2.1	1
	5.1	3.7	...	1.8	-4.2	9.3	1



Document	x						y
Don't buy this MI 6 Pro, Speaker volume is very bad	1.9	3.2	...	-9.8	-6.7	1.2	negative
Delivered as shown. Good price and fits perfect	1.3	3.6	...	-5.4	9.1	2.3	positive
What a phone.. A handy epic phone. MI at its best ...	0.4	7.6	...	-0.1	-1.4	8.7	positive
Its look stunning in pictures, but not in real.	1.5	-0.8	...	7.8	8.4	0.3	negative
Amazing camera and battery. Good deal!	2.5	-5.7	...	0.9	5.3	-8.1	positive



Data Curation

Data Augmentation

- Data augmentation is an integral process in deep learning, as in deep learning we need large amounts of data and in some cases it is not feasible to collect thousands or millions of images, so data augmentation comes to the rescue.
- It helps us to increase the size of the dataset and introduce variability in the dataset.
- Data augmentation is the process of increasing the amount and diversity of data.
- We do not collect new data, rather we transform the already present data.

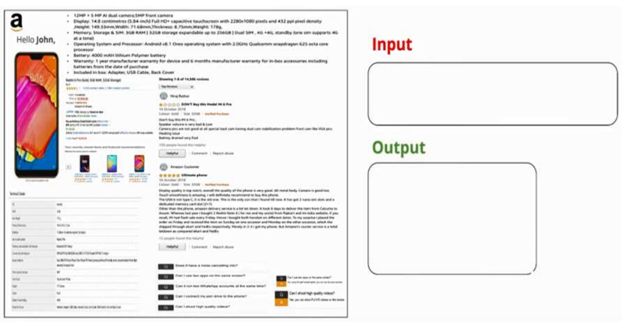
Operations in Data Augmentation

1) **Rotation**: Rotation operation as the name suggests, just rotates the image by a certain specified degree.
2) **Zooming:** Zooming operation allows us to either zoom in or zoom out.
3) **Cropping:** Cropping allows us to crop the image or select a particular area from an image.
4) **Flipping:** Flipping allows us to flip the orientation of the image. We can use horizontal or vertical flip.
5) **Changing the brightness level:** This feature helps us to combat illumination changes.

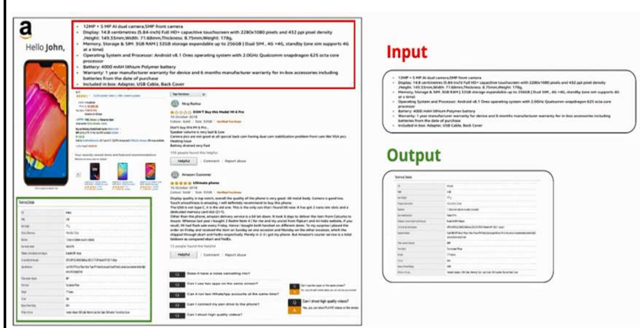
   You can encounter a scenario where most of your dataset comprises of images having a similar brightness level e.g. collecting the images of employees entering the office, by augmenting the images we make sure that our model is robust and is able to detect the person even in different surroundings.

What you want to do with data?

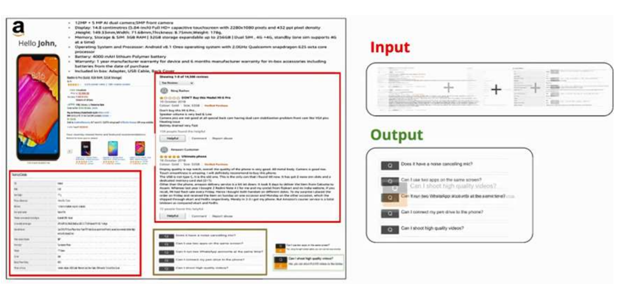
Task



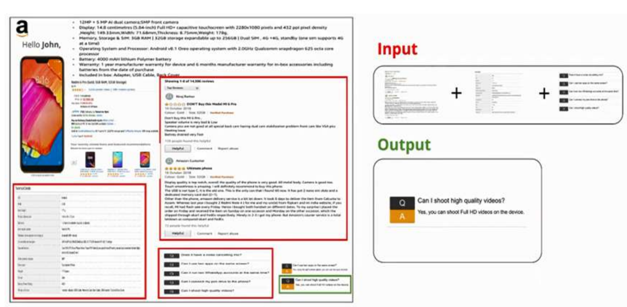
Task *(From description to structure specification)*



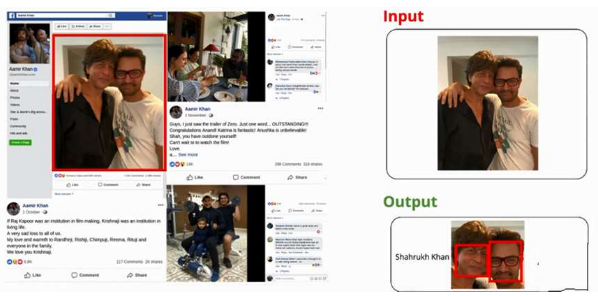
Task (From description + review to write FAQs)



Task *(From description + review + FAQs to Question Answering)*



Task (From image identify people)



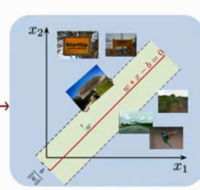
Task (From image identify activity)

Task (Classification)

Supervised

x					y
-8.5	-1.7	...	9.0	7.2	1
-0.4	6.7	...	4.7	-7.2	0
3.2	5.9	...	11.0	8.9	1
2.7	3.1	...	-2.1	9.7	0
3.9	7.8	...	-5.1	3.7	0
7.1	0.9	...	1.5	-4.2	1

Classification

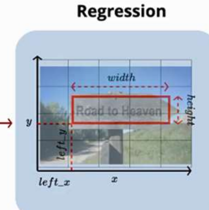


Task (Regression)

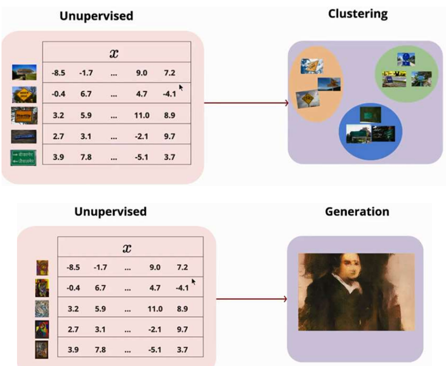
Supervised

x					$left\_x$	$left\_y$	$width$	$height$
-8.5	-1.7	...	9.0	7.2	2.3	1.2	9.2	10.1
0.9	-2.1	...	-8.1	1.9	4.3	4.2	7.1	5.1
2.9	-4.5	...	-3.7	8.9	2.3	7.2	6.9	7.3

Regression



Task *(Clustering)*



What is the mathematical formulation of a task?

Models

x y: bat, car, dog, cat, ship

$[\,2.1, 1.2, \ldots, 5.6, 7.2\,]$ $[\,0, 0, 1, 0, 0\,]$

$y \;= f(x)$ [true relation, unknown]

x y: bat, car, dog, cat, ship

$$\begin{bmatrix} 2.1, 1.2, \ldots, 5.6, 7.2 \\ 0.1, 3.1, \ldots, 1.7, 3.4 \\ 0.5, 9.1, \ldots, 5.1, 0.8 \\ 1.2, 4.1, \ldots, 6.3, 7.4 \\ 3.2, 2.1, \ldots, 3.1, 0.9 \end{bmatrix} \quad \begin{bmatrix} 0, 0, 1, 0, 0 \\ 0, 1, 0, 0, 0 \\ 0, 0, 0, 0, 1 \\ 1, 0, 0, 0, 0 \\ 0, 0, 1, 0, 0 \end{bmatrix}$$

$y \;= f(x)$ [true relation, unknown]

$\hat{y} \;= \hat{f}(x)$ [our approximation]

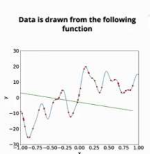
What are the choice of $\hat{f}$?

Models

$\hat{y} \;= \hat{f}(x)$ [our approximation]

$\hat{y} \;= mx + c$

$$\begin{array}{cc} x & y \\ \begin{bmatrix} 0.5 \\ 0.2 \\ 0.6 \\ \ldots \\ 0.3 \end{bmatrix} & \begin{bmatrix} 14.8 \\ 13.3 \\ 11.6 \\ \ldots \\ 6.16 \end{bmatrix} \end{array}$$

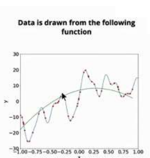


$\hat{y} \;= \hat{f}(x)$ [our approximation]

$\hat{y} \;= mx + c$

$\hat{y} \;= ax^2 + bx + c$

$$\begin{array}{cc} x & y \\ \begin{bmatrix} 0.5 \\ 0.2 \\ 0.6 \\ \ldots \\ 0.3 \end{bmatrix} & \begin{bmatrix} 14.8 \\ 13.3 \\ 11.6 \\ \ldots \\ 6.16 \end{bmatrix} \end{array}$$



What are the choice of $\hat{f}$?

Models

$\hat{y} \;= \hat{f}(x)$ [our approximation]

$$\begin{array}{cc} x & y \\ \begin{bmatrix} 0.5 \\ 0.2 \\ 0.6 \\ \ldots \\ 0.3 \end{bmatrix} & \begin{bmatrix} 14.8 \\ 13.3 \\ 11.6 \\ \ldots \\ 6.16 \end{bmatrix} \end{array}$$

$\hat{y} \;= mx + c$

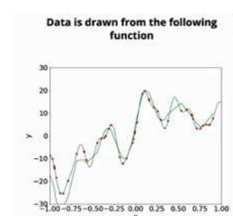
$\hat{y} \;= ax^2 + bx + c$

$\hat{y} \;= ax^3 + bx^2 + cx + d$

$\hat{y} \;= ax^4 + bx^3 + cx + d$

$\vdots$

$\hat{y} \;= ax^{25} + bx^{24} + \cdots + cx + d$

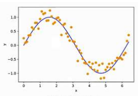


How do we know which model is better?

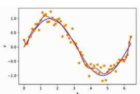
Loss function

- A loss function is a function that compares the target and predicted output values; measures how well the neural network models the training data. When training, we aim to minimize this loss between the predicted and target outputs.

$$\begin{array}{ccc} & x & y \\ 1 & 0.00 & 0.24 \\ 2 & 0.10 & 0.08 \\ 3 & 0.20 & 0.12 \\ .... & .... & .... \\ n & 6.40 & 0.36 \end{array}$$


True Function

$$\begin{array}{cccc} & x & y & \hat{f}_1(x) \\ 1 & 0.00 & 0.24 & 0.25 \\ 2 & 0.10 & 0.08 & 0.09 \\ 3 & 0.20 & 0.12 & 0.11 \\ .... & .... & .... & .... \\ n & 6.40 & 0.36 & 0.36 \end{array}$$

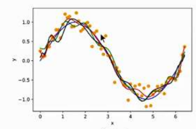

$\hat{f}_1(x)$

$\hat{f}_1(x) = 1.79x^{25} - 4.54x^{24} + ... - 1.48x + 2.48$

How do we know which model is better?

Loss function

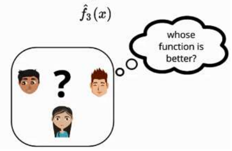
$$\begin{array}{cccccc} & x & y & \hat{f}_1(x) & \hat{f}_2(x) & \hat{f}_3(x) \\ 1 & 0.00 & 0.24 & 0.25 & 0.32 & 0.08 \\ 2 & 0.10 & 0.08 & 0.09 & 0.30 & 0.20 \\ 3 & 0.20 & 0.12 & 0.11 & 0.31 & 0.14 \\ .... & .... & .... & .... & .... & .... \\ n & 6.40 & 0.36 & 0.36 & 0.22 & 0.15 \end{array}$$


$\hat{f}_3(x)$

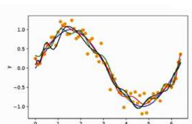
$\hat{f}_1(x) = 1.79x^{25} - 4.54x^{24} + ... - 1.48x + 2.48$

$\hat{f}_2(x) = 2.27x^{25} + 9.89x^{24} + ... + 2.79x + 3.22$

$\hat{f}_3(x) = 3.78x^{25} + 1.57x^{24} + ... + 1.01x + 8.68$



$$\begin{array}{cccccc} & x & y & \hat{f}_1(x) & \hat{f}_2(x) & \hat{f}_3(x) \\ 1 & 0.00 & 0.24 & 0.25 & 0.32 & 0.08 \\ 2 & 0.10 & 0.08 & 0.09 & 0.30 & 0.20 \\ 3 & 0.20 & 0.12 & 0.11 & 0.31 & 0.14 \\ .... & .... & .... & .... & .... & .... \\ n & 6.40 & 0.36 & 0.36 & 0.22 & 0.15 \end{array}$$


$\hat{f}_3(x)$

$\hat{f}_1(x) = 1.79x^{25} - 4.54x^{24} + ... - 1.48x + 2.48$

$\hat{f}_2(x) = 2.27x^{25} + 9.89x^{24} + ... + 2.79x + 3.22$

$\hat{f}_3(x) = 3.78x^{25} + 1.57x^{24} + ... + 1.01x + 8.68$

$\mathscr{L}_1 = \sum_{i=1}^{n}(y_i - \hat{f}_1(x_i))^2$

$\mathscr{L}_2 = \sum_{i=1}^{n}(y_i - \hat{f}_2(x_i))^2$

- *Square Error Loss*
- *Cross Entropy Loss*
- *KL divergence*

$$\begin{array}{cccccc} & x & y & \hat{f}_1(x) & \hat{f}_2(x) & \hat{f}_3(x) \\ 1 & 0.00 & 0.24 & 0.25 & 0.32 & 0.08 \\ 2 & 0.10 & 0.08 & 0.09 & 0.30 & 0.20 \\ 3 & 0.20 & 0.12 & 0.11 & 0.31 & 0.14 \\ .... & .... & .... & .... & .... & .... \\ n & 6.40 & 0.36 & 0.36 & 0.22 & 0.15 \end{array}$$

$$\begin{aligned} \mathscr{L}_1 &= \sum_{i=1}^{n}(y_i - \hat{f}_1(x_i))^2 \\ &= (0.24-0.25)^2 + (0.08-0.09)^2 + (0.12-0.11)^2 + \\ &\quad ... + (0.36-0.36)^2 \\ &= 1.38 \end{aligned}$$

$\mathscr{L}_1 = \sum_{i=1}^{n}(y_i - \hat{f}_1(x_i))^2 = 1.38$

$\mathscr{L}_2 = \sum_{i=1}^{n}(y_i - \hat{f}_2(x_i))^2 = 2.02$

$\mathscr{L}_3 = \sum_{i=1}^{n}(y_i - \hat{f}_3(x_i))^2 = 2.34$

How do we identify the parameter of the model?

Learning Algorithm (Optimizer)

- The weights and bias are modified using a function called Optimization Function.
- Optimization functions usually calculate the gradient i.e. the **partial derivative of loss function with respect to weights, and the weights** are modified in the opposite direction of the calculated gradient. This cycle is repeated until we reach the minima of loss function.

Budget in (100crs)	Box Office Collection in (100 crs)	Action Scene in times (100 mins)	IMDB Rating
0.55	0.66	0.22	4.8
0.68	0.91	0.77	7.2
0.66	0.88	0.67	6.7
⋮	⋮	⋮	⋮
0.72	0.94	0.97	8.1
0.58	0.74	0.35	5.3

$$\hat{f}(x) = ax_1^2 + bx_2^3 + cx_3^2$$

$$\hat{f}(x) = 3.5x_1^2 + 2.5x_2^3 + 1.2x_3^2$$

$$\mathscr{L}_1 = \sum_{i=1}^{n} (y_i - \hat{f}(x_i))^2$$

$$\hat{f}(x) = ax_1^2 + bx_2^3 + cx_3^2$$

Budget (100crore)	Box Office Collection(100 crore)	Action Scene times (100 mins)	IMDB Rating
0.55	0.66	0.22	4.8
0.68	0.91	0.77	7,2
0.66	0.88	0.67	6.7
⋮	⋮	⋮	⋮
0.72	0.94	0.97	8.1
0.58	0.74	0.35	5.3

a	b	c	Loss	Min Loss
20.0	20.0	-10.0	3.74	3.74

$$\mathscr{L}_1 = \sum_{i=1}^{n} (y_i - \hat{f}(x_i))^2$$

✗ In practice, brute force search is infeasible

$$\hat{f}(x) = ax_1^2 + bx_2^3 + cx_3^2$$

Budget (100crore)	Box Office Collection(100 crore)	Action Scene times (100 mins)	IMDB Rating
0.55	0.66	0.22	4.8
0.68	0.91	0.77	7,2
0.66	0.88	0.67	6.7
⋮	⋮	⋮	⋮
0.72	0.94	0.97	8.1
0.58	0.74	0.35	5.3

- ✓ Gradient Descent ++
- ✓ Adagrad
- ✓ RMSProp
- ✓ Adam

$$min_{a,b,c} \ \mathscr{L}_1 = \sum_{i=1}^{n} (y_i - \hat{f}(x_i))^2$$

✓ Many optimization solvers are available

How do we score for our model?

Evaluation

Top - 1

	True Labels	Predicted Labels	
$[\,2.1, 1.2, \ldots, 5.6, 7.8\,]$	1	1	✓
$[\,3.5, 6.6, \ldots, 2.5, 6.3\,]$	2	3	✗
$[\,6.3, 2.6, \ldots, 4.5, 3.8\,]$	3	1	✗
$[\,2.8, 3.6, \ldots, 7.5, 2.1\,]$	4	4	✓
$[\,2.2, 1.7, \ldots, 2.5, 1.8\,]$	5	5	✓
$[\,6.3, 2.6, \ldots, 4.5, 3.8\,]$	3	2	✗
$[\,1.9, 3.3, \ldots, 4.2, 1.1\,]$	5	5	✓

Class Labels	
Lion	1
Tiger	2
Cat	3
Giraffe	4
Dog	5

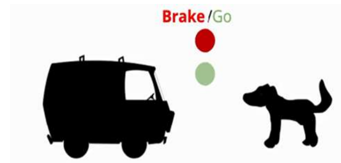
$$Accuracy = \frac{\text{Number of correct predictions}}{\text{Total number of predictions}} = \frac{4}{7} = 0.55$$

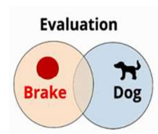
Top - 3

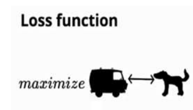
	True Labels	Predicted Labels	
$[\,2.1, 1.2, \ldots, 5.6, 7.8\,]$	1	$[\,1, 2, 3\,]$	✓
$[\,3.5, 6.6, \ldots, 2.5, 6.3\,]$	2	$[\,1, 2, 3\,]$	✓
$[\,6.3, 2.6, \ldots, 4.5, 3.8\,]$	3	$[\,1, 2, 3\,]$	✓
$[\,2.8, 3.6, \ldots, 7.5, 2.1\,]$	4	$[\,4, 5, 3\,]$	✓
$[\,2.2, 1.7, \ldots, 2.5, 1.8\,]$	5	$[\,5, 2, 1\,]$	✓
$[\,6.3, 2.6, \ldots, 4.5, 3.8\,]$	3	$[\,2, 1, 4\,]$	✗
$[\,1.9, 3.3, \ldots, 4.2, 1.1\,]$	5	$[\,5, 4, 1\,]$	✓

$$Accuracy = \frac{\text{Number of correct predictions in top-3}}{\text{Total number of predictions}} = \frac{6}{7} = 0.86$$

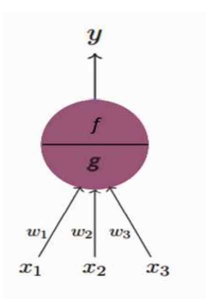
How Evaluation is different from loss function?

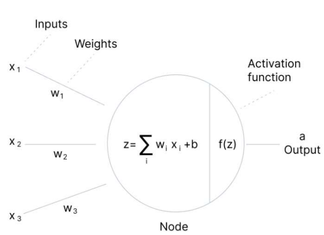






Artificial Neural Network





Common Terminology in ANN

Sample – A single row of a dataset.

Epoch – The number of times the algorithm runs on the whole training dataset.

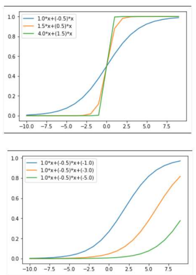
Batch – It denotes the number of samples to be taken to for updating the model parameters.

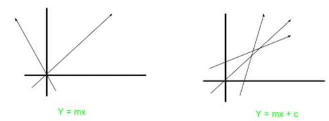
Learning rate – It is a parameter that provides the model a scale of how much model weights should be updated. Value is between 0-1.

Cost Function/Loss Function – A cost function is used to calculate the cost that is the difference between the predicted value and the actual value.

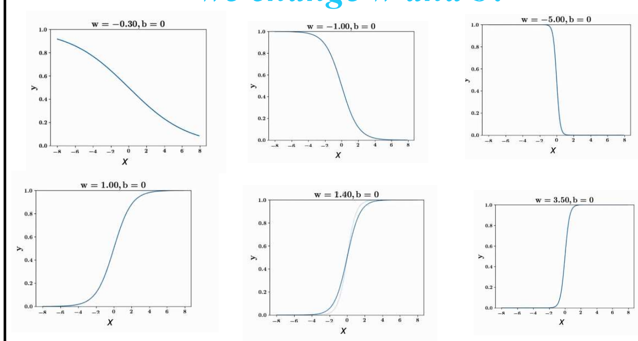
Weights/ Bias – The learnable parameters in a model that controls the signal between two neurons.

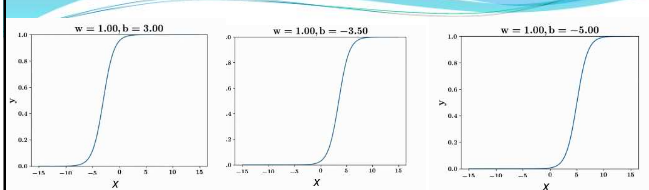
- Weight **increases the steepness** of activation function. This means weight decide how fast the activation function will trigger whereas bias is used to **delay the initiating of the** activation function.
- Due to absence of bias, model will train over point passing through origin only, which is not in accordance with real-world scenario. Also with the introduction of bias, the model will become more flexible.





How does the function behave when we change w and b?





1. w: (controls the slope)
   a. Negative w, negative slope, mirrored s-shape, becomes more harsh(vertical/less smooth) the more negative it goes
   b. Positive w, positive slope, normal s-shape, becomes more harsh(vertical/less smooth) the more positive it goes.

2. b: (controls the midpoint)

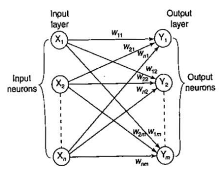
Connections

- The arrangement of neurons to form layers and the connection pattern formed within and between layers is called the network architecture.

1) Feed forward network: If no neuron in the output layer is an input to a node in the same layer / proceeding layer.
2) Feedback network: If outputs are directed back as input to the processing elements in the same layer/proceeding layer.
3) Lateral feedback: If the output is directed back to the input of the same layer.
4) Recurrent networks: Networks with feedback networks with closed loop

- There exist five basic types of connection architecture.
  1) Single layer feed forward network
  2) Multilayer feed-forward network
  3) Single node with its own feedback
  4) Single-layer recurrent network
  5) Multilayer recurrent network

Single layer feed forward network: In this type of network, we have only two layers input layer and output layer but the input layer does not count because no computation is performed in this layer. The output layer is formed when different weights are applied on input nodes and the cumulative effect per node is taken. After this, the neurons collectively give the output layer to compute the output signals.

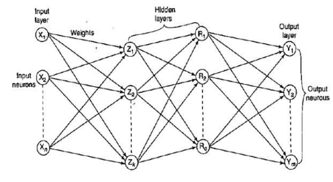


Single-layer feed-forward network

198

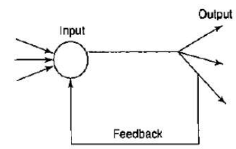
Multilayer feed-forward network: This network is formed by the interconnection of several layers. Input layer receives input and buffers input signal. Output layer generated output.

Layer between input and output is called hidden layer. Hidden layer is internal to the network. There are Zero to several hidden layers in a network. **More the hidden layer** more is the complexity of network, but efficient output is produced.



Multilayer feed-forward network

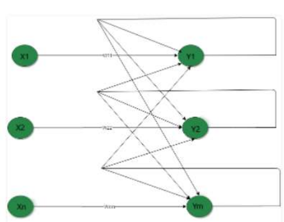
Single node with its own feedback: When outputs can be directed back as inputs to the same layer or preceding layer nodes, then it results in feedback networks. Recurrent networks are feedback networks with closed loops.



Single node with own feedback

Single-layer recurrent network: This network is a single-layer network with a feedback connection in which the processing element's output can be directed back to itself or to another processing element or both.

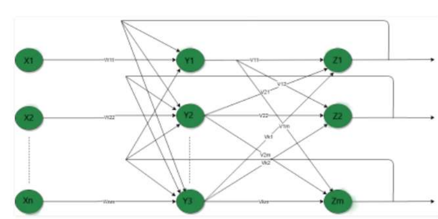
This allows it to exhibit dynamic temporal behavior for a **time sequence**. Unlike feedforward neural networks, RNNs can use their internal state (memory) to process sequences of inputs.



200

Multi-layer recurrent network: In this network, processing element output can be directed to the processing element in the same layer and in the preceding layer forming a multilayer recurrent network.

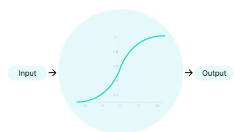
They perform the same task for every element of a sequence, with the output being dependent on the previous computations. Inputs are not needed at each time step. The main feature of a Recurrent Neural Network is its hidden state, which captures some information about a sequence.



201

Activation Function

- It will help to determine whether the neuron will **fire or not**.
- An activation function is **to add non-linearity** to the neural network.
- If we have a neural network working **without the activation** functions. In that case, **every neuron will only be performing a linear transformation** on the inputs using the weights and biases. It's because it **doesn't matter how many hidden layers we attach in the neural network**; all layers will behave in the same way because the composition of two linear functions is a linear function itself.



Activation Function

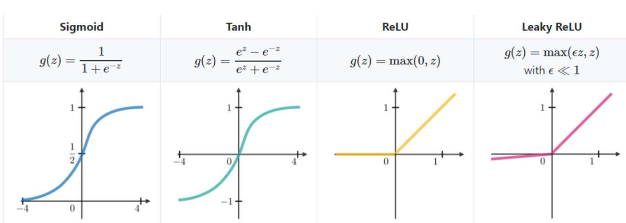
Activation Function Types

1) Linear Function
2) Binary Step Function
3) Non-Linear Function

Linear Function

- The linear activation function is also known as *Identity Function* where the **activation is proportional to the input**.

Non-linear Activation function



Softmax Function

- The softmax function is a function that turns a **vector of K real values into a vector of K real values that sum to 1.**
- The input values can be positive, negative, zero, or greater than one, but the softmax transforms them into values between 0 and 1, so that they can be interpreted as probabilities.
- If one of the inputs is small or negative, the softmax turns it into a small probability, and if an input is large, then it turns it into a large probability, but it will always remain between 0 and 1.

$$\sigma(\vec{z})_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}$$

$$\begin{bmatrix} 8 \\ 5 \\ 0 \end{bmatrix} \quad \begin{aligned} e^{z_1} &= e^8 = 2981.0 \\ e^{z_2} &= e^5 = 148.4 \\ e^{z_3} &= e^0 = 1.0 \end{aligned}$$

$$\sum_{j=1}^{K} e^{z_j} = e^{z_1} + e^{z_2} + e^{z_3} = 2981.0 + 148.4 + 1.0 = 3130.4$$

$$\sigma(\vec{z})_1 = \frac{2981.0}{3130.4} = 0.9523$$

$$\sigma(\vec{z})_2 = \frac{148.4}{3130.4} = 0.0474$$

$$\sigma(\vec{z})_3 = \frac{1.0}{3130.4} = 0.0003$$

Choosing the Right One

1) Use the ReLu or LeakyRelu function in hidden layers only
2) In binary classification remember to use the Sigmoid function in the output layer
3) In multi-class classification(when classes to predict are more than 2) problem use Softmax function in the output layer
4) Due to the vanishing gradient problem ‘Sigmoid’ and ‘Tanh’ activation functions are avoided sometimes in deep neural network architectures
5) Always remember you can also invent your **own activation** function and can check its performance.

Learning Rules

- Learning rule or Learning process is a method or a mathematical logic. It improves the Artificial Neural Network’s performance and applies this rule over the network.
- Thus learning rules updates the weights and bias levels of a network when a network simulates in a specific data environment.

1) **Hebbian learning**
2) **Perceptron learning rule**
3) **Delta learning**
4) **Correlation learning rule**
5) **Outstar learning rule**

Hebbian Learning Rule

- The Hebbian rule was the first learning rule. In 1949 Donald Hebb developed it as learning algorithm of the neural network.
- The Hebb learning rule assumes that – If two neighbor neurons **activated and deactivated** at the same time. Then the weight connecting these neurons should **increase.**
- For neurons operating in the **opposite phase**, the weight between them should **decrease**.
- If there is no signal correlation, the weight **should not change**.

$$w_i(new) = w_i\,(old) + x_i\, y$$

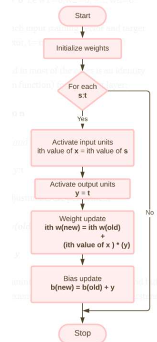
- When inputs of both the nodes are either **positive or negative**, then a **strong positive weight exists** between the nodes.
- If the input of a node is **positive and negative** for other, a strong **negative weight exists** between the nodes.
- At the start, values of all **weights are set to zero**. This learning rule can be used for both soft- and hard-activation functions.

- Hebbs network is suited more for **bipolar data**. If binary data is used, the weight updation formula cannot distinguish two conditions namely:

1. A training pair in which an input unit is "on" and the target value is "off".
2. A training pair in which both the input unit and the target value is "off".

Flow Diagram

s: t refers to each training input and target output pair. Till there exists a pair of training input and target output, the training process takes place; else it is stopped.



STEP 1:Initialize the weights and bias to **'0'** i.e w1=0,w2=0, …., wn=0.

STEP 2: 3–5 have to be performed for each input training vector and target output pair **i.e. s:t** (s=training input vector, t=training output vector)

STEP 3: Input units activation are set and in most of the cases is an identity function(one of the types of an activation function) for the input layer;

$x_i = s_i$ **for i=1 to n**

Identity Function:*Its a linear function and defined as* **f(x)=x for all x**

STEP 4: Output units activations are set y:t

STEP 5: Weight adjustments and bias adjustments are performed;

$w_i(new) = w_i\ (old) + (x_i * y)$

$bias(new) = bias(old) + y$

Implementation of AND Gate with Hebbian Learning Rule

Set all weights to zero, $\mathbf{w_i = 0}$ for i=1 to n, and **bias to zero**.

	INPUT			TARGET	
	x_1	x_2	b		y
X_1	-1	-1	1	Y_1	-1
X_2	-1	1	1	Y_2	-1
X_3	1	-1	1	Y_3	-1
X_4	1	1	1	Y_4	1

$w_i\ (new) = w_i\ (old) + x_i\ y$

$b\ (new) = b\ (old) + y$

Activation function used here is Bipolar Sigmoidal Function so the range is [-1,1].

Step 1 :

Set weight and bias to zero, w = [0 0 0]^T and b = 0.

Step 2 :

Set input vector $X_i = S_i$ for i = 1 to 4.

X_1 = [-1 -1 1]^T

X_2 = [-1 1 1]^T

X_3 = [1 -1 1]^T

X_4 = [1 1 1]^T

Step 3 :

Output value is set to y = t.

Inputs				Weight Changes			Weights		
x1	x2	b	y	Δw1	Δw2	Δb	w1 (0	w2 0	b 0)
1	1	1	1						
1	-1	1	-1						
-1	1	1	-1						
-1	-1	1	-1						

Step 4 :

Modifying weights using Hebbian Rule:

$$\Delta w_1 = x_1 y = 1 \times 1 = 1$$
$$\Delta w_2 = x_2 y = 1 \times 1 = 1$$
$$\Delta b = y = 1$$

$$w_i(\text{new}) = w_i(\text{old}) + x_i y$$
$$w_1(\text{new}) = w_1(\text{old}) + x_1 y = 0 + 1 \times 1 = 1$$
$$w_2(\text{new}) = w_2(\text{old}) + x_2 y = 0 + 1 \times 1 = 1$$
$$b(\text{new}) = b(\text{old}) + y = 0 + 1 = 1$$

The weight change here is

$$\Delta w_1 = x_1 y = 1 \times -1 = -1$$
$$\Delta w_2 = x_2 y = -1 \times -1 = 1$$
$$\Delta b = y = -1$$

The new weights here are

$$w_1(\text{new}) = w_1(\text{old}) + \Delta w_1 = 1 - 1 = 0$$
$$w_2(\text{new}) = w_2(\text{old}) + \Delta w_2 = 1 + 1 = 2$$
$$b(\text{new}) = b(\text{old}) + \Delta b = 1 - 1 = 0$$

Inputs			Weight Changes			Weights		
x1	x2	y	Δw1	Δw2	Δb	w1 (0	w2 0	b 0)
1	1	1	1	1	1	1	1	1
1	-1	-1	-1	1	-1	0	2	0
-1	1	-1	1	-1	-1	1	1	-1
-1	-1	-1	1	1	-1	2	2	-2

The separating line equation is given by

$$x_2 = \frac{-w_1}{w_2} x_1 - \frac{b}{w_2}$$

For all inputs, use the final weights obtained for each input to obtain the separating line. For the first input [1 1 1], the separating line is given by

$$x_2 = \frac{-1}{1} x_1 - \frac{1}{1} \Rightarrow x_2 = -x_1 - 1$$

Similarly, for the second input [1 −1 1], the separating line is

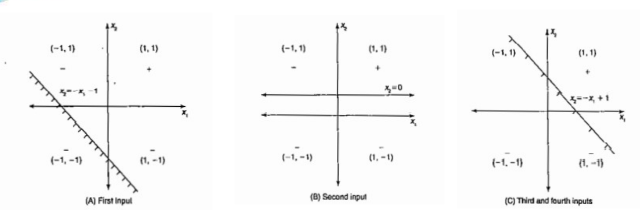
$$x_2 = \frac{-0}{2} x_1 - \frac{0}{2} \Rightarrow x_2 = 0$$

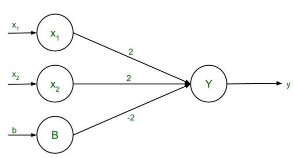
For the third input [−1 1 1], it is

$$x_2 = \frac{-1}{1} x_1 + \frac{1}{1} \Rightarrow x_2 = -x_1 + 1$$

Finally, for the fourth input [−1 −1 1], the separating line is

$$x_2 = \frac{-2}{2} x_1 + \frac{2}{2} \Rightarrow x_2 = -x_1 + 1$$





Perceptron Learning Rule

- This rule is an **error correcting** and a **supervised learning algorithm** of single **layer feedforward networks with linear activation function**.
- As being supervised in nature, to calculate the error, there would be a comparison between the desired/target output and the actual output.
- If there is any difference found, then a change must be made to the weights of connection.
- The perceptron network consists of three units, namely, sensory unit (input unit), associator unit (hidden unit), response unit (output unit).
- The input units are connected to hidden units with fixed weights having values **1, 0 or -l,** which are assigned at random.
- **The binary activation** function is used in hidden layer.
- The response unit has an activation of **l, 0 or -1**

The output of the perceptron network is given by

$$y = f(y_{in})$$

where $f(y_{in})$ is activation function and is defined as

$$f(y_{in}) = \begin{cases} 1 & if\, y_{in} > \theta \\ 0 & if -\theta \le y_{in} \le \theta \\ -1 & if\, y_{in} < -\theta \end{cases}$$

If $y \neq t$, then

$$w_i(new) = w_i(old) + \alpha\, t x_i$$

$$b(new) = b(old) + \alpha\, t$$

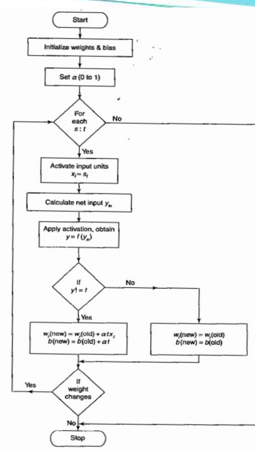
"t" is +1 or-l and α is the learning rate.

else,

$$w_i(new) = w_i(old)$$

$$b(new) = b(old)$$

- If no error occurs, there is no weight updation and hence the training process may be stopped.



Step 0: Initialize the weights and the bias (for easy calculation they can be set to zero). Also initialize the learning rate $\alpha(0 < \alpha \le 1)$. For simplicity α is set to 1.

Step 1: Perform Steps 2–6 until the final stopping condition is false.

Step 2: Perform Steps 3–5 for each training pair indicated by $s{:}t$.

Step 3: The input layer containing input units is applied with identity activation functions:

$$x_i = s_i$$

Step 4: Calculate the output of the network. To do so, first obtain the net input:

$$y_{in} = b + \sum_{i=1}^{n} x_i w_i$$

where "n" is the number of input neurons in the input layer. Then apply activations over the net input calculated to obtain the output:

$$y = f(y_{in}) = \begin{cases} 1 & \text{if } y_{in} > \theta \\ 0 & \text{if } -\theta \le y_{in} \le \theta \\ -1 & \text{if } y_{in} < -\theta \end{cases}$$

Step 5: *Weight and bias adjustment:* Compare the value of the actual (calculated) output and desired (target) output.

If $y \neq t$, then

$$w_i(\text{new}) = w_i(\text{old}) + \alpha t x_i$$
$$b(\text{new}) = b(\text{old}) + \alpha t$$

else, we have

$$w_i(\text{new}) = w_i(\text{old})$$
$$b(\text{new}) = b(\text{old})$$

Step 6: Train the network until there is no weight change. This is the stopping condition for the network. If this condition is not met, then start again from Step 2.

Perceptron Summary

- It required initial weight to be assigned random values.
- Uses supervised learning
- Guarantees convergence in finite iterations
- Used in feedforward network

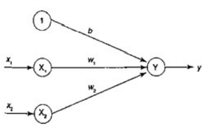
Implement AND function using perceptron networks for bipolar inputs ans targets.

x_1	x_2	t
1	1	1
1	−1	−1
−1	1	−1
−1	−1	−1

$$y = f(y_{in}) = \begin{cases} 1 & \text{if } y_{in} > 0 \\ 0 & \text{if } y_{in} = 0 \\ -1 & \text{if } y_{in} < 0 \end{cases}$$

$$\Delta w_1 = \alpha t x_1;$$
$$\Delta w_2 = \alpha t x_2;$$
$$\Delta b = \alpha t$$

$w_1 = w_2 = b = 0$ and $\theta = 0$. The learning rate α is set equal to 1.



$$w_i(\text{new}) = w_i(\text{old}) + \alpha t x_i$$
$$w_1(\text{new}) = w_1(\text{old}) + \alpha t x_1 = 0 + 1 \times 1 \times 1 = 1$$
$$w_2(\text{new}) = w_2(\text{old}) + \alpha t x_2 = 0 + 1 \times 1 \times 1 = 1$$
$$b(\text{new}) = b(\text{old}) + \alpha t = 0 + 1 \times 1 = 1$$

- Calculate the net input

$$y_{in} = b + x_1 w_1 + x_2 w_2$$
$$= 0 + 1 \times 0 + 1 \times 0 = 0$$

Input			Target	Net input	Calculated output	Weight changes			Weights		
x_1	x_2	1	(t)	(y_{in})	(y)	Δw_1	Δw_2	Δb	w_1 (0	w_2 0	b 0)
EPOCH-1											
1	1	1	1								
1	−1	1	−1								
−1	1	1	−1								
−1	−1	1	−1								

Input			Target	Net input	Calculated output	Weight changes			Weights		
x_1	x_2	1	(t)	(y_{in})	(y)	Δw_1	Δw_2	Δb	w_1 (0	w_2 0	b 0)
EPOCH-1											
1	1	1	1	0	0	1	1	1	1	1	1
1	−1	1	−1	1	1	−1	1	−1	0	2	0
−1	1	1	−1	2	1	+1	−1	−1	1	1	−1
−1	−1	1	−1	−3	−1	0	0	0	1	1	−1
EPOCH-2											
1	1	1	1	1	1	0	0	0	1	1	−1
1	−1	1	−1	−1	−1	0	0	0	1	1	−1
−1	1	1	−1	−1	−1	0	0	0	1	1	−1
−1	−1	1	−1	−3	−1	0	0	0	1	1	−1

The final weights and bias after second epoch are

$$w_1 = 1, w_2 = 1, b = -1$$

Since the threshold for the problem is zero, the equation of the separating line is

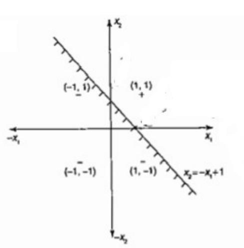
$$x_2 = -\frac{w_1}{w_2} x_1 - \frac{b}{w_2}$$

Here

$$w_1 x_1 + w_2 x_2 + b > \theta$$
$$w_1 x_1 + w_2 x_2 + b > 0$$

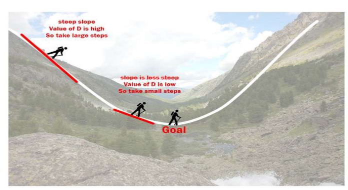
Thus, using the final weights we obtain

$$x_2 = -\frac{1}{1} x_1 - \frac{(-1)}{1}$$
$$x_2 = -x_1 + 1$$



Gradient Descent

- Gradient descent is an iterative optimization algorithm to find the minimum of a function. Here that function is our Loss Function.
- Loss function is the partial deviation with respect to w and b.



- He goes down the slope and takes large steps when the slope is steep and small steps when the slope is less steep.
- He decides his next position based on his current position and stops when he gets to the bottom of the valley which was his goal.

Delta Learning Rule (Widrow−Hoff Rule)

- The perceptron learning rule originates from the Hebbian assumption while the delta rule is derived from the gradient- descent method.
- The delta rule updates the weights between the connections so as to minimize the **difference between the net input to the output unit and the target value.**
- The major aim is to minimize all errors over all training patterns. This is done by reducing the error for each pattern, one at a time.
- Delta rule (DR) is similar to the Perceptron Learning Rule (PLR), with some differences:
  1. Error (Error (δ), in DR is not restricted to having values of 0, 1, or - 1 (as in PLR), but may have any value
  2. DR can be derived for any differentiable output/activation function f, whereas in PLR only works for threshold output function.

Case-I − when **t ≠ y**, then

$$w(new) = w(old) + \Delta w$$

where

$$\Delta w_i = \eta(t - o)x_i$$

learning rate, target value, perceptron output, input value

Case-II − when **t = y**, then

No change in weight

$$E(w_{ij}) = \frac{1}{2}(y_{tarj} - y_j)^2 \qquad y_j = f(a_j) = \sum_i w_{ij} x_i$$

$$\frac{\partial E}{\partial w_{ij}} = \frac{\partial E}{\partial y_j} \cdot \frac{\partial y_j}{\partial w_{ij}} = -(y_{tarj} - y_j)x_i = -\delta x_i$$

$$w_i(\text{new}) = w_i(\text{old}) + \alpha\,(t - y_{in})\,x_i$$
$$b(\text{new}) = b(\text{old}) + \alpha\,(t - y_{in})$$